December 23, 1992

Air Traffic Control Flow Management Automation Final Report

Bill Nedell

A report summarizing the development of the NASA Center/TRACON Automation System.

NAG 2-537

This report will summarize the development of a prototype air traffic control automation system using advanced color workstation technology. The report begins with a background discussion of previous work done by NASA in airborne guidance and trajectory synthesis leading to the development of an air traffic control system based on the concept of providing ground based trajectory analysis for each aircraft in the system.

The general requirements for developing a prototype system will be discussed with emphasis on the lessons learned in developing full featured, "useful" prototype systems.

1.0 Background

The section discusses previous work done by NASA leading to the development of a prototype Center/TRACON Automation System, or CTAS. The first attempt at developing a prototype on a mini computer architecture will be described. The current workstation platform capabilities will be described.

1.1 Previous work

Prior to 1986 NASA had developed a real-time Air Traffic Control (ATC) simulation based on a mini computer architecture. This system was used to conduct experiments in advanced air traffic control automation tools using ground based airborne guidance augmentation concepts developed at NASA over a number of years.The system was comprised of a Xerox Sigma 9 computer running a real time operating system, and an Evans and Sutherland graphics system driven by a Digital Equipment Corporation PDP 11-34 computer. The graphics system supported two color, vector drawn displays which were used as ATC controller positions. In addition to the graphics displays several ANSI terminals were used to control system execution and to control the flight of computer driven aircraft in the target generation system used to support the ATC experiments.

While the system provided the capability to run simulation experiments of reasonable complexity the system was difficult to program and the computational resources (storage and speed) were seriously taxed. A decision was made to upgrade the system to provide the means to expand the fidelity and flexibility of the experimental system to allow the development of a research prototype with significantly greater storage capacity and processing speed. Also required was the ability to expand the system to include more display positions with better graphics capabilities for both the simulated controller positions as well as the target generation control positions.

1.2 Workstation Platform

During this period workstation technology had been exploding. Processing speed, disk and RAM storage had increased in quantity and dropped in price significantly. The engineering workstation was rapidly replacing the central computer as the system of choice for many engineering applications. These workstations generally provided color graphics capabilities that offered good performance and were relatively easy to program. Other features becoming common were the Unix operating system and its suite of productivity tools. High speed networks were generally incorporated making it easy to connect many workstations together to exchange data and share resources.

NASA procured two Sun Microsystems workstations to explore migrating the ATC simulation into a distributed workstation environment. The workstation technology seemed particularly well suited to the ATC automation tool development application. Rather than having a limited number of several special purpose displays connected at a single mainframe or mini computer, a system based on a distributed workstation environment would allow any number of displays to be used with the functionality of each station being determined by the software that was executed on it. The same system could be used as an ATC control position, target generator control, system execution and monitoring. The same position could also be used for development greatly increasing the efficiency of the development process. Distributing the software over several computers greatly increased the computational performance available for each task. It was also easy to expand the system scope by simply procuring additional workstations. Distributing the software also provided the framework in which to maintain functional divisions amongst the processes thus keeping the software modules relatively clean, easy to understand and maintain.

The workstation vendor was chosen for their good support software including color graphics and windowing capability, their open systems philosophy, and their commitment to remain a leader in cost/performance ratio. This decision has payed out nicely over the years. We have gone from a 0.5 MIPS (million instruction per second) Sun 2 to a 40 MIPS SPARCstation 2 in six years. Each upgrade in hardware has been accomplished without changing any of the code. The same applies to upgrades to the operating system which have been accomplished with a minimum of development time. Sun has been challenged on a number of fronts but continue to provide excellent capabilities and continue to be a leader in the ongoing development of the open systems approach.

Table 1 below gives an indication of the rapid progress made in the engineering workstation computer category using a real world example of two systems procured by NASA separated by a period of ten years.

TABLE 1. You've Come A Long Way, Baby!

	LSI 11/23 – 1979	SPARCstation 1 – 1989
Cost (U.S. $)	$20,000	$20,000
RAM	128 kilo-bytes (maximum)	16 mega-bytes (typical)
DISK (typical)	2 x 10 mega-byte (removable platters)	300 mega-bytes
Operating System	RSX-11M	Sun OS (Unix based)
Graphics	none	high resolution color graphics
Performance	130,000 instructions per second	12 million instructions per second

2.0 General requirements

After procuring the initial Sun workstations several requirements were identified for the system port from the mini environment to the workstation environment. Many of the requirements evolved along with the development of the system. While the original system was intended only as a lab prototype it was decided to try and make the new system as useful and full featured as possible.

2.1 Useful prototype.

In developing the initial layout of the CTAS system it very quickly become obvious that is would be relatively easy to create a system that had a high degree of fidelity compared with operational ATC platforms. For example, in order to create the video maps that are used to draw the plan view map display it was decided to use data from operational ATC facilities rather than hand creating a more limited set of graphical features sufficient for a lab based system. This was done and in the long run saved significant amounts of time when it became necessary to keep the displays up to date with the operational facilities.

This theme was found to be repeated over and over in the course of the development. Any attempt to shortcut the path by a implementing a quick fix was found in the long run to be counter productive. It was always easier to do it right the first time than to fix it later. On the other hand it is important to recognize that the CTAS development is a research system and as such is subject to constant revision. Attempts too complete in the early stages could be counter productive as well as features may be found to be inadequate and need to be redone. Thus some kind of balance needs to be struck in what I will term a "useful prototype". A useful prototype is easiest to describe in context. In the CTAS several different tools have been developed and are undergoing refinement by continued simulation testing as well as field evaluations. The evaluation conducted dur-

ing this process can only be carried out if the fidelity of the system being evaluated is good enough that the subjects using the system are not led to false conclusions by the limitations of the tool's implementation. Furthermore to gain long term experience the system to needs to be mature enough that it can be used in field sites by operational personnel. The system needs to be able to handle all the configurations and situations that arise in the real world environment for extended periods of time. This real world experience is then fed back into the system design resulting in improvements which are then further refined in the field. A too complicated project structure would prevent timely incorporation of these improvements. A useful prototype is a system that is meant to strike a balance between these conflicting requirements. The system is kept as simple as possible while still incorporating the features necessary to make it work in the real world environment. Several characteristics necessary to achieve this balance are described below.

2.2 Remain Flexible.

In any project of reasonable complexity there are going to be numerous issues that arise that were not anticipated in the original design. Too rigid a design will make it difficult or impossible to incorporate the changes necessary without significant delays. In many ways the "top down" design model does not work in complex, computer based system prototype development. There are too many unknowns at the outset. More emphasis needs to be placed on building in the capability to adjust to the unexpected.

2.3 Create expansion potential.

There is an adage that you can never have too much closet space. This also applies to computer system resources. Virtually every parameter will be found to stretch the limits of capacity. In the early stages of the CTAS project the 0.5 Mips Sun 3s were thought to process adequate performance for the current system requirements plus some expansion potential. Today the 40 Mips SPARC 2s seem inadequate to the task and we anxiously await the advent of the next generation of machines. The same phenomena applies to RAM and disk storage. This feature is not got to go away. The best one can do is anticipate the need for ever greater resources and to choose the hardware and software platforms that seem best able to evolve along with the requirements.

2.4 Benefit from new technologies.

In some ways this is an extension of the previous issue. Primarily this is accomplished by using commercial off-the-shelf technology whenever possible. It is very difficult to maintain a large research prototype that contains special purpose hardware and software since any changes to the research system typically imply an equivalent change to the special purpose products.

2.5 Be amenable to ad-hoc project management.

Typically in the early stages of the development of a research prototype there are many ideas being discussed by individuals involved in the project which are difficult to capture in exact language or describe in unequivocal terms. By definition a research project is one in which the issues being addressed are poorly understood. Meaningful solutions develope gradually and may require reworking the software modules several times

before practical solutions are found. Much of the knowledge gained during this period is passed on through word of mount. Although the CTAS research group was fairly small initially (four to five people) it was still fairly difficult to coordinate the activities of this group too tightly without stifling the creative process necessary to further the research goals. For this reason the computer system that embodies the research concepts needs to be accessible to the researchers without requiring the researcher to wade through several layers of software structure or management. For CTAS this meant creating simple coding standards that were easy to learn and apply. Use of source code control and automated program build were particularly important both in reducing the learning curve as well as enforcing a minimal discipline for new staff.

2.6 Provide maximum leverage of resources.

Advanced operating systems such as Sun's implementation of Unix generally include tools developed over time that can significantly reduce development time and enhance productivity. Many times features that are included in the computer platform or available off-the-shelf are overlooked and the proverbial wheel is reinvented. In the case of the Sun OS the tools provided include inter-process communication, networking, data base management, source code control, program build control tools, symbolic debugging, built in graphics including widget sets, event processing libraries, etc. The small amount of time invested learning to use these resources pays dividends in the long run.

2.7 Build broad foundations from the start.

Even though the CTAS system described in this report is intended as a research prototype and not an operational system it is just as important to provide solid foundations on which to build up and improve the system. Trying to take shortcuts or creating poorly designed structures makes it difficult or impossible to improve those later in the project. Figure 1 is meant to illustrate this point graphically. Many software initially make rapid

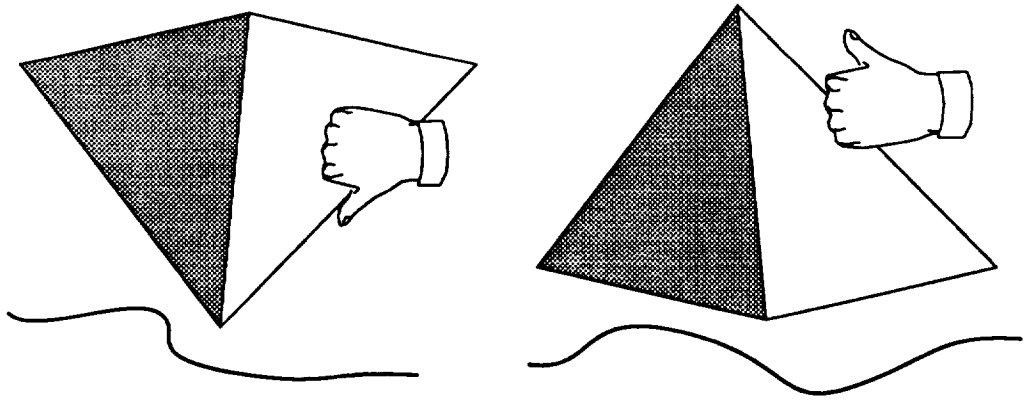

FIGURE 1. There is a right way and a wrong way.

progress but then get bogged down. I believe this is frequently because of a lack of attention to the structural foundation of the software. It is frequently tempting to take shortcuts to develop specific capabilities as fast as possible but this approach is sure to cause problems in the future years of the project. Conversely any (perceived) extra effort spent in the early stages of a project creating general structures that can be built on in the later stages of the project usually pays out handsomely in increased productivity through the advanced stages.

An article in the Spring 1992 issue of *SunProgrammer* identified two characteristics of good software. One is conciseness and the other is leverage. Conciseness is important for several reasons. Concise code is generally easier to learn and modify. Usually code written in a concise fashion is also efficient. It is possible to program in a cryptic fashion which may result in concise code that is not easy to maintain but this is not generally true. This is because writing concisely general requires more thought into the construction of the code and its flow. It is hard to write concise code at the first attempt. Writing concisely generally means going over the code several times continuously improving it, weeding out inefficient constructs, seeing global patterns that allow a more general construction, etc. In this way it is very similar to writing prose. It is accepted that in prose it is necessary to review the text and revise it a number of times in order to improve it and make it clearer. Why not apply the same criteria to software?

The article quotes Newton as saying, "If I have seen farther than others, it is because I have stood on the shoulders of giants." We have already discussed one aspect of leverage: using system resources to their fullest. Building broad structures also provides a sort of built in leverage. If written correctly, many aspects of the software being developed should be reusable in many different ways. In some ways this is related to the conciseness issue in that it is hard to write code that is easily reusable the first time through. It general takes going over the code several times refining it at each pass. Patterns become obvious in hindsight and the code can be rewritten in ways that allow the common elements to be combined and reused. This theme of revisiting code and making constant improvements is a very important aspect of software development. Most useful, long lived, widely distributed pieces of code have usually been developed in this fashion. This concept is particularly applicable to research environments and leads to the conclusion discussed in the next section.

2.8 Top Down Design, Not!

Very rarely is a system truly designed in a top down approach, particularly in systems that are primarily of a research nature. Given a little thought this makes sense. Frequently software systems start out trying to address poorly understood problems that are felt to be amenable to some kind of computer based solution. An initial attempt is made at providing tools to address the concerns. Generally this first attempt generates a lot of additional ideas and discussion and provide insights that lead to further refinements of the system as well as possibly the development of entirely new concepts which were not obvious until a prototype or simulation of the system was available that could be interacted with. At this stage the system architecture and tool functionality can be refine or redone to incorporate the understanding gained from the first stage.

Whether the software needs to largely rewritten or can evolve to a more capable system largely depends on how the foundations were built in the earlier stages. It is generally

possible to anticipate overall needs of a system and to build in the expansion potential needed for the inevitable future growth. Certain portions of the software can possibly be identified at various stages whose functionality is understood well enough to separate out from the body of software still undergoing development. This code is now a candidate for a "top down design". But in essence the software has really already been designed and what is really being accomplished is top down *documentation*. Because the system has been developed and tested over a period of time the functionality required has developed through interaction with users and will therefore lead to a robust top down description. This description (design) then serves as a guide to the developers in accomplishing the necessary changes when rewriting the relevant portion of the software system.

2.9 Recognize the programmers.

The preceding discussion should make clear the important role the programmers have in the creation and evolution of a computer based software system. In the past when computer systems were less accessible and resources more limited it was common to have a staff dedicated to maintaining the computer and operating system and providing access to its resources. With the advent of workstations and the tremendous capabilities being placed on the desks of the engineer/developer has also come the added responsibilities of maintaining and using these greater resources. In essence the greater capabilities have led to being able to develope much more capable systems with fewer staff but the demands placed on this staff are also much greater.

The areas that the developers need to be proficient include general system software, graphics and windows, database, inter-process communication, real time operation, etc. This is very different from the days in which software tended to be more of a straight line, algorithmic approach with a programmer working directly for a researcher to implement solutions to scientific computation problems. Current software systems tend to be very complex with the software architecture itself being part of the design requirements. Frequently the researcher and programmer tend to be the same person. At the very least the researcher needs to familiar enough with software design issues to be able to work with a programmer to come up with feasible solutions to specific problems that take into account the complexities inherent in implementing, maintaining, and extending the software.

3.0 Conclusion

The development of operational prototype software has a unique set of requirements. This report has summarized the lessons learned in developing the CTAS software. At this juncture certain CTAS components have been deemed ready for operational development. Work is commencing on a "hardened" CTAS that will be capable of being used as an operational tool at ATC facilities.